REMARKS / ARGUMENTS

Claims 1-21 are now pending.

No claims stand allowed.

Claims 1-6, 8-13, and 15-20 have been amended to further particularly point out and distinctly claim subject matter regarded as the invention. The text of claims 7, 14, and 21 is unchanged, but their meaning is changed because they depend from amended claims. No "new matter" has been added by the amendment.

The drawings have been amended to correct minor errors noted in the Office Action and otherwise. These corrections are of a clerical nature and do not add "new matter".

Changes to the Drawings

The Examiner has requested corrected drawings corresponding to proposed substitute drawings filed on March 2, 2003.[1] The requested corrected drawings are submitted herewith.

Claim Objections

Claims 1-6, 8-13, and 15-20 stand objected to as allegedly lacking proper antecedent basis.[2] With this Amendment, claim 1 has been amended to insert

[1] Office Action dated May 19, 2003 ¶ 3.
[2] Office Action ¶ 4.

the word 'object-oriented' between "said" and "library" in the third line of claim 1. Claims 2-6, 8-13, and 15-20 have also been amended to insert the word 'object-oriented' between "said" and "library". Claim 3 has been amended to remove the first "in" from the phrase "classes and interfaces in defined in said (object-oriented) library" in the third line of claim 3. Corresponding changes have been made claims 10 and 17. Accordingly, the Applicant respectfully requests the objection to claims 1-6, 8-13, and 15-20 be withdrawn.

The First 35 U.S.C. §103 Rejection

Claims 1-6, 8-13, and 15-20 stand rejected under 35 U.S.C. §103(a) as being allegedly unpatentable over Fitzgerald[3] in view of Aizikowitz et al.[4] [5] This rejection is respectfully traversed.

According to M.P.E.P. §2143,

> To establish a *prima facie* case of obviousness, three basic criteria must be met. First there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art, not in the applicant's disclosure.

[3] USP 5,408,665.
[4] USP 6,526,571.
[5] Office Action ¶ 5.

Furthermore, the mere fact that references can be combined or modified does not render the resultant combination obvious unless the prior art also suggests the desirability of the combination. *In re Mills*, 916 F.2d 680, 16 USPQ2d 1430 (Fed. Cir. 1990).

Claim 1

Claim 1 as amended recites:

A method for representing an application programming interface (API) definition for an object-oriented library, said method comprising:
creating a list of public elements in said object-oriented library, each of said public elements including a sublist of all public hierarchically-related elements that are a parent of the element; and
storing said list.

The Examiner states:

Fitzgerald teaches a system and method for listing public elements in a library as claimed. See Figures 3-4 and the corresponding portions of Fitzgerald's specification for this disclosure. Refer also to claims 1 and 6 for more details of this disclosure. In particular, Fitzgerald teaches a method for representing an application programming interface (API) definition for a programming language library [260], said method comprising:
creating [Librarian 265 creates] a list [Standard Dictionary 360 (also 430): 'a list of the library's public symbols and module names' (Column 8, lines 51-59)] of public elements [library object modules (See Fig. 313)] in said programming language library, each of said public elements including a sublist [Dependency List 445] of all public related elements for the element ['each module it needs' (Column 11, lines 17-25) See also Column 3, lines 13-25]; and storing [stored in Library File 410 (See Fig. 4A)] said list.

Fitzgerald does not explicitly state that the library (260) is an object-oriented library as claimed. However, Fitzgerald does state that in the preferred embodiment, the programming language specific to the system is Borland C++. See column 5, lines 46-59 for this disclosure. C++ being an object-oriented language, this provides direct suggestion that Fitzgerald's library is an object-oriented library as claimed. Furthermore, one can infer that Fitzgerald's library is object-oriented because it stores objects. See Figures 3B-4A and the corresponding potions of Fitzgerald's specification for this disclosure.
Aizikowitz teaches a system and method similar to that of Fitzgerald, wherein a class dependency hierarchy is generated from an object oriented library. See Figures 1 & 2 and the corresponding portions of Aizikowitz' specification for this

disclosure. In particular, Aizikowitz teaches the practice of creating a class hierarchy (CHG) for classes and interfaces of a Java package (object-oriented library).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to apply Fitzgerald's method of creating a list of public elements reflecting their dependencies to the object-oriented library of Aizikowitz in order to derive the object-oriented library dependency hierarchy in a list structure as claimed. One would have been motivated to do so because of the suggestions provided by Fitzgerald as above.[6]

The Applicant respectfully disagrees for the reasons set forth below.

I. Fitzgerald and Aizikowitz et al. Do Not Teach All Claim Limitations.

When evaluating a claim for determining obviousness, all limitations of the claim must be evaluated.[7]

Contrary to the Examiner's statement, Fitzgerald does not disclose object-oriented libraries. The Examiner attempts to equate the term "object module" used in Fitzgerald with the term "object-oriented" of claim 1. The two terms are entirely different. As explained in Fitzgerald,

> The ultimate output of the compiler is an "object module". Although an object module includes object code for instructing the operation of a computer, the object module is not in a form which may be directly executed by a computer. Instead, it must undergo a "linking" operation before the final executable program is created. While linking involves many specific operations, it may be thought of as the general process of combining or linking together one or more compiled object modules to create an executable program. This task usually falls to a "linker." In typical operation, a linker receives, either from the user or from an integrated compiler, a list of object modules desired to be included in the link operation. The object modules themselves may either be stored separately in standalone object or .OBJ files which the compiler has generated, or in "library"

[6] Office Action ¶ 5.
[7] *In re Dillon*, 919 F.2d 688, 16 USPQ2d 1897 (Fed. Cir. 1990).

> or .LIB files which are .OBJ files aggregated into a single file by a "librarian". In operation, the linker scans the object modules from the object and library files specified. After resolving interconnecting references as needed, the linker constructs an executable image by organizing the object code from the modules of the program in a format understood by the operating system program loader.[8]

Thus, a library in Fitzgerald is a file containing an aggregation of files containing object *code*.

In object-oriented programming, an object is an entity that has state, behavior and identity. An object's state consists of its attributes and the attributes' current values. An object's behavior consists of the operations that can be performed on it and the accompanying state changes. In traditional procedural languages, code and data are *separate*. In the object-oriented approach, code and data that belong together can be *combined* into objects.

The Applicant submits the Examiner's suggestion that because the compiler disclosed in Fitzgerald is a C++ compiler, the library disclosed in Fitzgerald is object-oriented, is improper. First, while the C++ language includes object-oriented features, the language can be used to write programs that do not use the object-oriented features of the language. Second, the Examiner has not shown why a compiler for a language that supports object-oriented features must use a library that itself is object-oriented.

[8] Fitzgerald at col. 2 lines 1-25.

The Applicant also submits the Examiner's statement that one can infer Fitzgerald's library is object-oriented because it stores objects is improper. Contrary to the Examiner's statement, Fitzgerald does not disclose storing objects as disclosed in the Application and claimed in claim 1. As mentioned above, Fitzgerald stores object *files* containing object code. The Examiner is reminded that the mere absence from a reference of an explicit requirement of a claim cannot be reasonably construed as an affirmative statement that the requirement is in the reference.[9]

Furthermore, the cited references do not disclose or suggest creating a list of public elements in an object-oriented library, where each of the public elements include a sublist of all public *hierarchically-related* elements *that are a parent* of the element. Fitzgerald discloses a dependency list containing "each module it needs", such as modules *called*. With this Amendment, claim 1 has been amended to make this distinction more clear. The amendment to claim 1 finds support in the Specification at p. 11 line 4 – p. 12 line 22, p. 16 lines 12-20, and FIGS. 6B-6C.

For the above reasons, the 35 U.S.C. § 103 rejection is unsupported by the art. Thus, no prima facie case of obviousness has been established and the 35 U.S.C. § 103 rejection should be withdrawn.

[9] *In re Evanega*, 829 F.2d 1110, 4 USPQ2d 1249 (Fed. Cir. 1987).

II. There Is No Basis in the Art for Combining or Modifying Fitzgerald

MPEP § 2143 provides:

> The mere fact that references *can* be combined or modified does not render the resultant combination obvious unless the prior art also suggests the desirability of the combination.[10] (emphasis added)

Furthermore,

> Obviousness cannot be established by combining the teachings of the prior art to produce the claimed invention, absent some teaching, suggestion or incentive supporting the combination.[11]

Regarding the motivation to combine Fitzgerald and Aizikowitz et al., the Examiner contends:

> One would have been motivated to do so because of the suggestions provided by Fitzgerald as above.[12]

The Applicants submit the reasons for the combination of references suggested by the Examiner do not constitute particular findings as required by the Federal Circuit. The Federal Circuit has stated:

> The motivation, suggestion or teaching may come explicitly from statements in the prior art, the knowledge of one of ordinary skill in the art, or, in some cases the nature of the problem to be solved. In addition, the teaching, motivation or suggestion may be implicit from the prior art as a whole, rather than expressly stated in the references. ... The test for an implicit showing is what the combined teachings, knowledge of one of ordinary skill in the art, and the nature of the

[10] *In re Mills*, 916 F.2d 680, 16 USPQ2d 1430 (Fed. Cir. 1990).
[11] *ACS Hospital Systems, Inc. v. Monteffore Hospital*, 732 F.2d 1572, 1577, 221 USPQ 929, 933 (Fed. Cir. 1984).
[12] Office Action ¶ 5.

> problem to be solved as a whole would have suggested to those of ordinary skill in the art. ... Whether the Board relies on an express or an implicit showing, it must provide particular findings related thereto. Broad conclusory statements standing alone are not "evidence."[13]

The Examiner's broad conclusory statement that "One would have been motivated to do so because of the suggestions provided by Fitzgerald as above" standing alone is not evidence.

For this additional reason, the 35 U.S.C. § 103 rejection is unsupported by the art. Thus, no prima facie case of obviousness has been established and the 35 U.S.C. § 103 rejection should be withdrawn.

III. Neither Fitzgerald Nor Aizikowitz et al. Teaches The Problem Or Its Source.

Fitzgerald is directed to a system and method for efficient linking of object files. More particularly, Fitzgerald discloses linking object modules from .OBJ files and library files, where library object modules which are not needed for the link may be determined before the libraries are scanned during the first pass of the linker, thus allowing library object modules which are not needed during subsequent linking operations to be skipped.[14]

[13] *In re Kotzab*, 217 F.3d 1370, 55 USPQ2d 1317 (Fed. Cir. 2000) (citations omitted).
[14] Fitzgerald Abstract.

Aizikowitz et al. is directed to a system and method for identifying calls in a Java package whose targets are guaranteed to belong to the package. More particularly, Aizikowitz et al. discloses determining inheritance graph and access permissions of respective components in the package, both of which are used in combination with the knowledge that the package is sealed and signed to determine whether all the targets of a call are guaranteed to belong to the package.[15]

Whereas the art of the present invention concerns representing an application programming interface (API) in an object-oriented system such that submerged hierarchies are enabled. Neither Fitzgerald nor Aizikowitz et al. recognized the need for an API representation that sufficiently constrains particular implementations, while allowing them to define submerged hierarchies. Therefore, the Applicant submits there would be no motivation to combine the teachings of Fitzgerald with the teachings of Aizikowitz et al.

For this additional reason, the 35 U.S.C. § 103 rejection is unsupported by the art. Thus, no prima facie case of obviousness has been established and the 35 U.S.C. § 103 rejection should be withdrawn.

[15] Aizikowitz et al. Abstract.

Dependent Claim 2

Claim 2 depends from claim 1 and thus include the limitations of claim 1. The argument set forth above is equally applicable here. The base claim being allowable, the dependent claim must also be allowable at least for the same reasons.

Claims 3 and 4

Claims 3 and 4 include limitations similar to claims 1 and 2. Claims 1 and 2 being allowable, claims 3 and 4 must also be allowable.

Claim 5

Claim 5 as amended recites:

A method for determining a program hierarchy, said method comprising:
receiving an application programming interface (API) definition file for an object-oriented library, said API definition file including a list of public elements in said object-oriented library, each of said public elements including a sublist of all public hierarchically-related elements that are a parent of the element; and
indicating a first public element is a direct parent of a second public element when said first public element is represented in the sublist for said second public element and said first public element is not represented in the sublist for any other public element listed in the sublist for said second public element.

The Examiner states:

the system and method of Fitzgerald in view of Aizikowitz as applied to claim 1 above discloses the invention as claimed. See Figure 6 and the corresponding portion of Fitzgerald's specification for this disclosure. In particular, Fitzgerald (as modified by Aizikowitz) teaches a method for determining a program hierarchy, said method comprising:
receiving [Step 601] an application programming interface (API) definition file [Standard and Extended Dictionaries] for an object-oriented library, said API definition file including ...[See the discussion regarding claim 1 above]; and traversing the program hierarchy through the dependency list [See Fig. 6C].
Fitzgerald (as modified by Aizikowitz) does not explicitly teach the step of

"indicating a first public element is a direct parent of a second public element" as claimed. However, looking at the structure of Fitzgerald's (as modified by Aizikowitz) Extended Dictionary described above with regard to claims 1 and 2, one can infer that the direct parent of a specific module (public element) is represented in the sublist (dependency list) of that module, but is not represented in the sublist of any other modules listed in that module's sublist. In other words, in order to traverse Fitzgerald's (as modified by Aizikowitz) hierarchy, a first module's direct parent can be found by searching that first module's sublist to find the second module that is not listed in the sublist for any other module in the first module's sublist. It would have been obvious to one of ordinary skill in the art at the time the invention was made to program Fitzgerald's (as modified by Aizikowitz) system to traverse the Extended Dictionary's hierarchy to find a first module's direct parent by searching that first module's sublist to find the second module that is not listed in the sublist for any other module in the first module's sublist as claimed. One would have been motivated to do so because this method is easily inferred from the structure of the Extended Dictionary, and seems to be the only method for traversing the hierarchy possible.[16]

The arguments made with respect to claim 1 apply here as well. Additionally, the Applicant submits the Examiner has impermissibly engaged in hindsight construction. The Federal Circuit has repeatedly warned against the use of hindsight construction:

> A critical step in analyzing the patentability of claims pursuant to 103(a) is casting the mind back to the time of invention, to consider the thinking of one of ordinary skill in the art, guided only by the prior art references and the then-accepted wisdom in the field. ... Close adherence to this methodology is especially important in cases where the very ease with which the invention can be understood may prompt one "to fall victim to the insidious effect of a hindsight syndrome wherein that which only the invention taught is used against its teacher."[17]

Additionally, the M.P.E.P. states:

> The examiner must step backward in time and into the shoes worn by the hypothetical 'person of ordinary skill in the art' when the invention was unknown and just before it was made ... the examiner must put aside knowledge of the applicant's disclosure, refrain from using hindsight, and consider the subject matter claimed "as a whole."[18]

[16] Office Action ¶ 5.
[17] *In re Kotzab*, 217 F.3d 1369, 55 USPQ2d 1316 (Fed. Cir. 2000) (citations omitted).
[18] *Id.*

The Examiner's statement that "one would have been motivated to do so because this method is easily inferred from the structure of the Extended Dictionary, and seems to be the only method for traversing the hierarchy" indicates the Examiner failed to step backward in time and into the shoes worn by a person of ordinary skill in the art when the invention was unknown and just before it was made in determining whether the Applicant's invention is rendered obvious under 35 U.S.C. § 103. This is hindsight construction.

Furthermore, as mentioned above, Fitzgerald's extended dictionary includes all modules needed by an element, including modules related via a calling relationship. Aizikowitz discloses a hierarchy, which includes both parents and children of an element. Combining both would result in a sublist that includes public elements related to a particular element via a calling relationship, a parent relationship or a child relationship. As such, no inference could be made regarding whether a first public element is a direct parent of a second public element by examining sublists as claimed in claim 5.

For the above reasons, the Applicants submit no prima facie case of obviousness has been established and the 35 U.S.C. § 103 rejection should be withdrawn.

Claim 6

The arguments made above with respect to claims 2 and 5 apply here. Claims 2 and 5 being allowable, claim 6 must also be allowable.

Claims 8-13

Claims 8-13 are In re Beauregard claims corresponding to method claims 1-6. The argument set forth above is equally applicable here. Claims 1-6 being allowable, claims 8-13 must also be allowable.

Claims 15-20

Claims 15-20 are means-plus-function claims corresponding to method claims 1-6. The argument set forth above is equally applicable here. Claims 1-6 being allowable, claims 15-20 must also be allowable.

Accordingly, it is respectfully requested that the rejection of claims based on Fitzgerald in view of Aizikowitz et al. be withdrawn. In view of the foregoing, it is respectfully asserted that the claims are now in condition for allowance.

The Second 35 U.S.C. §103 Rejection

Claims 7, 14 and 21 stand rejected under 35 U.S. C. 103(a) as being unpatentable over Fitzgerald in view of Aizikowitz et al. and further in view of Gossian et al.[19] [20] This rejection is respectfully traversed.

Claim 7

Claim 7 recites:

[19] USP 5,974,255.
[20] Office Action ¶ 6.

The method of claim 5, further comprising
comparing a first program hierarchy reconstructed from a first API definition file with a second program hierarchy reconstructed from a second API definition file; and
indicating an error when said first program hierarchy is inconsistent with said second program hierarchy.

The Examiner states:

the system and method of Fitzgerald in view of Aizikowitz as applied to claim 5 above does not explicitly disclose the steps of comparing two reconstructed program hierarchies and indicating an error when they are inconsistent as claimed. However, Aizikowitz does disclose the need to maintain integrity of the program hierarchy in order to maintain the signed and sealed status of the package. See the Background and Summary of the Invention sections of Aizikowitz' specification for this disclosure. This provides suggestion for examining the hierarchy of an API with an expected hierarchy to maintain consistency for the signed and sealed status. Gossain discloses a method for testing the inheritance hierarchy of an object-oriented class structure by comparing the active hierarchy to a test hierarchy stored within the system. See the Figure and the Detailed Description of the Drawing section for this disclosure. Refer specifically to column 3, lines 6-14. Gossain teaches the two claimed steps as follows:
Comparing [step 18] a first program hierarchy [hierarchy of class under test (11)] with a second program hierarchy [test class hierarchy (12)]; and
Indicating an error [Column 3, lines 9-10] when said first program hierarchy is inconsistent [`when a difference between the current state and expected state... is detected' (Column 3, lines 7-8)] with said second program hierarchy.
It would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate Gossain's method for testing class hierarchies into Fitzgerald's (as modified by Aizikowitz) system such that the system would compare the hierarchy reconstructed from an Extended Dictionary for one library with the hierarchy reconstructed from a test Extended Dictionary, and indicate an error when the two hierarchies were inconsistent. One would have been motivated to do so because of Aizikowitz' suggestion described above.[21]

The Applicants respectfully disagree. Claim 7 depends from claim 5. The base claim being allowable, the dependent claim must also be allowable. Furthermore, contrary to the Examiner's statement, the cited references do not disclose comparing a

[21] Office Action ¶ 6.

first program hierarchy *reconstructed from a first API definition file* with a second program hierarchy *reconstructed from a second API definition file*. There is nothing to reconstruct, neither in Fitzgerald as modified by Aizikowitz et al., nor in Gossain et al., since hierarchical information is already present in the form of a complete hierarchy. Whereas the hierarchical information is not immediately discernable in the method of independent claim 5 and dependent claim 7, thus requiring the program hierarchies to be reconstructed.

For this additional reason, the 35 U.S.C. § 103 rejection is unsupported by the art. Thus, no prima facie case of obviousness has been established and the 35 U.S.C. § 103 rejection should be withdrawn.

Claim 14

Claim 14 is an In re Beauregard claim corresponding to method claim 7. The argument set forth above is equally applicable here. Claim 7 being allowable, claim 14 must also be allowable.

Claim 21

Claim 21 is means-plus-function claim corresponding to method claim 7. The argument set forth above is equally applicable here. Claim 7 being allowable, claim 21 must also be allowable.

Accordingly, it is respectfully requested that the rejection of claims based on Fitzgerald in view of Aizikowitz et al. and further in view of Gossian et al. be withdrawn. In view of the foregoing, it is respectfully asserted that the claims are now in condition for allowance.

The Third 35 U.S.C. §103 Rejection

Claims 1-4, 8-11, and 15-18 stand rejected under 35 U.S.C. §103(a) as being allegedly unpatentable over Sweeney et al.[22] in view of Aizikowitz et al.[23] This rejection is respectfully traversed.

Claim 1

The Examiner states:

> Sweeney discloses a system and method for generating an object-oriented program inheritance listing. See Figures 3, 4 & 7 and the corresponding portions of Sweeney's specification for this disclosure. In particular, Sweeney teaches a method for representing an application programming interface (API) definition for an object-oriented program, said method comprising:
> creating [Steps 703-707] a list [Class Hierarchy (See Fig. 3 and column 3, line 59 - column 4, line 4)] of public elements [set of classes] in said object-oriented program, each of said public elements [`for every class' (column 4, line 2)] including a sublist of all public related elements [`the set of base classes it inherits from is specified' (column 4, lines 2-3)] for the element; and storing said list [See column 19, lines 56-62].
> Sweeney does not explicitly disclose that the object-oriented program used for generating the class hierarchy is an object-oriented library as claimed. However, Sweeney does disclose the use and importance of object-oriented libraries in the background of the invention section (See column 1, lines 11-24). This provides suggestion for applying Sweeney's method to an object-oriented library.
> Aizikowitz teaches a system and method similar to that of Sweeney, wherein a class dependency hierarchy is generated from an object oriented library. See Figures 1 & 2 and the corresponding portions of Aizikowitz' specification for this disclosure. In

[22] USP 6,230,314.
[23] Office Action ¶ 7.

particular, Aizikowitz teaches the practice of creating a class hierarchy (CHG) for classes and interfaces of a Java package (object-oriented library).
It would have been obvious to one of ordinary skill in the art at the time the invention was made to apply Sweeney's method of creating a list of public elements reflecting their dependencies to the object-oriented library of Aizikowitz in order to derive the object-oriented library's dependency hierarchy in a list structure as claimed. One would have been motivated to do so because of the suggestions provided by Sweeney as above.[24]

The Applicant respectfully disagrees for the reasons set forth below.

I. Sweeney et al. and Aizikowitz et al. Do Not Teach All Claim Limitations.

Contrary to the Examiner's statement, Sweeney et al. does not disclose creating a list of public elements in an object-oriented library, where each of the public elements include a sublist of *all* public related elements for the element. Sweeney et al. states:

> The inheritance relations among the classes; in particular, for every class, the set of base classes it inherits from is specified. The inheritance may be further qualified as virtual or non-virtual inheritance.
> As an example, FIG. 3 shows a C++ class hierarchy that consists of the classes S, A, B, C, and D having the following members:
> class S has (data) member y, virtual member (function) foo, and non-virtual member (function) bar,
> class A has (data) member x.
> Class B has virtual member (function) foo and has virtual member (function) bar,
> class C has non-virtual member (function) bar, and
> class D has no members of its own.
> Classes S,A,B,C, and D have the following inheritance relations:
> class B inherits nonvirtually from class A,
> class B inherits virtually from class S,
> class C inherits nonvirtually from class A, and inherits virtually from class S,
> class D inherits nonvirtually from class B, and inherits nonvirtually from class C.[25]

Figure 3 of Sweeney et al. is reproduced below.

[24] Office Action ¶ 7.
[25] Sweeney et al. col. 4 lines 1-23.

```
class S {
public:
    int y;
    virtual void foo ();
    void bar ();
};

class A {
public:
    int x;
};

class B : public A, vitual public S {
public:
    virtual void foo ();
};

class C : public A, vitual public s {
public:
    void bar ();
};

class D : public B, public C {
};
```

FIG. 3

As shown in FIG. 3 and the corresponding text of Sweeney et al., class D inherits from class B and class C, class C inherits from class A and class S, class B inherits from class A and class S. However, the set of base classes class D inherits from specifies only the direct parents of class D, specifically class B and class C. Thus, as is apparent from the above example in Sweeney et al., the "set of base classes it inherits from" refers to classes that are *direct* parents, not *all* public related elements.

Furthermore, the cited references do not disclose or suggest creating a list of public elements in an object-oriented library, where each of the public elements include a sublist of all public *hierarchically-related* elements *that are a parent* of the element. With this Amendment, claim 1 has been amended to make this distinction more clear. The amendment to claim 1 finds support in the Specification at p. 11 line 4 – p. 12 line 22, p. 16 lines 12-20, and FIGS. 6B-6C.

Additionally, the Examiner makes the following misstatement:

> Sweeney does not explicitly disclose that the object-oriented program used for generating the class hierarchy is an object-oriented library as claimed.[26]

Contrary to the Examiner's statement, claim 1 does not claim using an object-oriented program to generate a class hierarchy is an object-oriented library.

[26] Office Action ¶ 7.

For the above reasons, the 35 U.S.C. § 103 rejection is unsupported by the art. Thus, no prima facie case of obviousness has been established and the 35 U.S.C. § 103 rejection should be withdrawn.

II. There Is No Basis in the Art for Combining or Modifying Sweeney et al.

Regarding the motivation to combine Sweeney et al. and Aizikowitz et al., the Examiner contends:

> One would have been motivated to do so because of the suggestions provided by Sweeney as above.[27]

The Applicants submit the reasons for the combination of references suggested by the Examiner do not constitute particular findings as required by the Federal Circuit. The Federal Circuit has stated:

> The motivation, suggestion or teaching may come explicitly from statements in the prior art, the knowledge of one of ordinary skill in the art, or, in some cases the nature of the problem to be solved. In addition, the teaching, motivation or suggestion may be implicit from the prior art as a whole, rather than expressly stated in the references. ... The test for an implicit showing is what the combined teachings, knowledge of one of ordinary skill in the art, and the nature of the problem to be solved as a whole would have suggested to those of ordinary skill in the art. ... Whether the Board relies on an express or an implicit showing, it must provide particular findings related thereto. Broad conclusory statements standing alone are not "evidence."[28]

The Examiner's broad conclusory statement that "One would have been motivated to do so because of the suggestions provided by Sweeney as above" standing alone is not evidence.

[27] Office Action ¶ 7.
[28] *In re Kotzab*, 217 F.3d 1370, 55 USPQ2d 1317 (Fed. Cir. 2000) (citations omitted).

For this additional reason, the 35 U.S.C. § 103 rejection is unsupported by the art. Thus, no prima facie case of obviousness has been established and the 35 U.S.C. § 103 rejection should be withdrawn.

III. Neither Sweeney et al. Nor Aizikowitz et al. Teaches The Problem Or Its Source.

Sweeney et al. is directed to a mechanism that eliminates redundant components from objects of a program. More particularly, Sweeney et al. discloses detecting situations where a member of a given class is used by some, but not all instances of that class, and the elimination of this member from the instances where it is not needed. This is accomplished by an analysis of the program and its class hierarchy, followed by the construction of a new, specialized class hierarchy and a transformation of the program.[29]

Aizikowitz et al. is directed to a system and method for identifying calls in a Java package whose targets are guaranteed to belong to the package. More particularly, Aizikowitz et al. discloses determining inheritance graph and access permissions of respective components in the package, both of which are used in combination with the knowledge that the package is sealed and signed to determine whether all the targets of a call are guaranteed to belong to the package.[30]

[29] Sweeney et al. Abstract.
[30] Aizikowitz et al. Abstract.

Whereas the art of the present invention concerns representing an application programming interface (API) in an object-oriented system such that submerged hierarchies are enabled. Neither Sweeney et al. nor Aizikowitz et al. recognized the need for an API representation that sufficiently constrains particular implementations, while allowing them to define submerged hierarchies. Therefore, the Applicant submits there would be no motivation to combine the teachings of Sweeney et al. with the teachings of Aizikowitz et al.

For this additional reason, the 35 U.S.C. § 103 rejection is unsupported by the art. Thus, no prima facie case of obviousness has been established and the 35 U.S.C. § 103 rejection should be withdrawn.

Dependent Claim 2

Claim 2 depends from claim 1 and thus include the limitations of claim 1. The argument set forth above is equally applicable here. The base claim being allowable, the dependent claim must also be allowable at least for the same reasons.

Claims 3 and 4

Claims 3 and 4 include limitations similar to claims 1 and 2. Claims 1 and 2 being allowable, claims 3 and 4 must also be allowable.

Claims 8-11

Claims 8-11 are In re Beauregard claims corresponding to method claims 1-4.

The argument set forth above is equally applicable here. Claims 1-4 being allowable, claims 8-11 must also be allowable.

Claims 15-18

Claims 15-18 are means-plus-function claims corresponding to method claims 1-4. The argument set forth above is equally applicable here. Claims 1-4 being allowable, claims 15-18 must also be allowable.

Accordingly, it is respectfully requested that the rejection of claims based on Sweeney et al. in view of Aizikowitz et al. be withdrawn. In view of the foregoing, it is respectfully asserted that the claims are now in condition for allowance.

In view of the foregoing, it is respectfully asserted that the claims are now in condition for allowance.

Request for Allowance

It is believed that this Amendment places the above-identified patent application into condition for allowance. Early favorable consideration of this Amendment is earnestly solicited.

Request for Entry of Amendment

Entry of this Amendment will place the Application either in condition for allowance, or at least, in better form for appeal by narrowing any issues. Accordingly, entry of this Amendment is appropriate and is respectfully requested.

If, in the opinion of the Examiner, an interview would expedite the prosecution of this application, the Examiner is invited to call the undersigned attorney at the number indicated below.

Respectfully submitted,
THELEN REID & PRIEST, LLP

[illegible]

Dated: September 19, 2003

John P. Schaub
Reg. No. 42,125

Thelen Reid & Priest LLP
P.O. Box 640640
San Jose, CA 95164-0640
Tel. (408) 292-5800
Fax. (408) 287-8040